This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS

- TEXT CUT OFF AT TOP, BOTTOM OR SIDES

- FADED TEXT

- ILLEGIBLE TEXT

- SKEWED/SLANTED IMAGES

- COLORED PHOTOS

- BLACK OR VERY BLACK AND WHITE DARK PHOTOS

- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problems Mailbox.
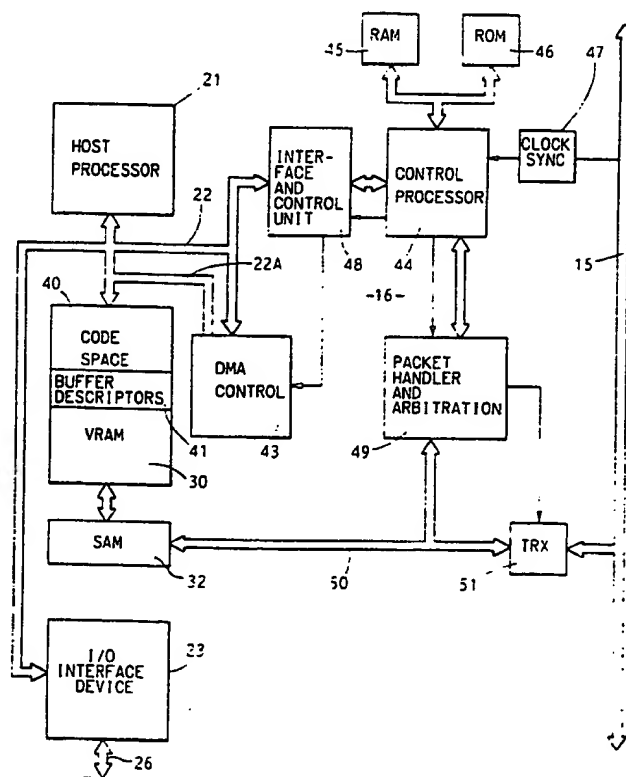
THIS PAGE BLANK (USPTO)

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 13/12, 15/16	A1	(11) International Publication Number: WO 91/10958
		(43) International Publication Date: 25 July 1991 (25.07.91)

(21) International Application Number: PCT/GB91/00035

(22) International Filing Date: 11 January 1991 (11.01.91)

(30) Priority data:
9000643.8   11 January 1990 (11.01.90)   GB

(71) Applicant (for all designated States except US): RACAL-MILGO LIMITED [GB/GB]; Landata House, Station Road, Hook, Hampshire RG27 9PE (GB).

(72) Inventors; and
(75) Inventors/Applicants (for US only) : BOAL, John, Hill [GB/GB]; 2 Sleepersdelle Gardens, Off Sparkford Road, Winchester, Hampshire SO22 4NU (GB). NEWTON, Mark, Kendal [GB/GB]; 17 Caraway Road, Earley, Reading, Berkshire RG6 2XR (GB).

(74) Agent: ABNETT, Richard, Charles; Reddie & Grose, 16 Theobalds Road, London WC1X 8PL (GB).

(81) Designated States: AT, AT (European patent), AU, BB, BE (European patent), BF (OAPI patent), BG, BJ (OAPI patent), BR, CA, CF (OAPI patent), CG (OAPI patent), CH, CH (European patent), CM (OAPI patent), DE, DE (European patent), DK, DK (European patent), ES, ES (European patent), FI, FR (European patent), GA (OAPI patent), GB, GB (European patent), GR (European patent), HU, IT (European patent), JP, KP, KR, LK, LU, LU (European patent), MC, MG, ML (OAPI patent), MR (OAPI patent), MW, NL, NL (European patent), NO, PL, RO, SD, SE, SE (European patent), SN (OAPI patent), SU, TD (OAPI patent), TG (OAPI patent), US.

Published
With international search report.

(54) Title: COMPUTER BUS SYSTEM

(57) Abstract

Data transfers between an input/output device (23) and a packet bus (15) are effected with minimum intervention by the host processor (21). To this end a dual port VRAM (30) has one port connected by a bus (22) to the device (23) and the other port connected via a bus (50) and packet bus controller (16) to the packet bus (15). A serial access memory (32) effects word serial/parallel conversion between 1 kbyte buffers in the VRAM (30) and 16 bit words on the bus (15). The packet bus controller has an associated host-system interface (48) for controlling transfers effected by the packet bus controller. The packet bus controller (16) "mirrors" the I/O interface device (23) so that data transfer can take place transparently through the VRAM (30).

COMPUTER BUS SYSTEM

Technical Field of the Invention

The present invention relates to a computer bus system. In one aspect the invention concerns a system wherein a plurality of computer systems communicate via a bus. In another aspect the invention concerns a system comprising a host computer, a peripheral controller, a communications bus and means for transferring data between the peripheral controller and the bus.

The invention is particularly, but not exclusively, concerned with a system in which the bus is a backplane bus for connecting a plurality of computing systems, referred to herein as "engines", typically by way of the backplane of a rack carrying the engines on respective boards.

Background to the Invention

A backplane bus differs from a microprocessor bus in that it has to be more rigorously specified to handle the loading of a plurality of boards and the extra transmission distances via the backplane. Moreover arbitration control may well be required. Standard backplane buses transfer information through shared bus memory. Each system has a bus controller between the system memory and the bus. Peripheral controllers are well known devices used to manage the flow of data to and from peripheral units of all kinds. Some controllers are very simple but others such as disc controllers are of considerable complexity. Another example of a complex controller is a local area network (LAN) coprocessor such as that sold by Intel Corporation under the type number 82596. Peripheral controllers are designed to relieve the host processor of detailed control operations. For example the LAN coprocessor referred to above interfaces the host processor to a LAN and manages memory structures automatically with DMA control. Nevertheless the host processor necessarily has a complex driver written in conformity with the control requirements of the coprocessor, just as it has to have drivers for a disc controller, and so on.

Regardless of the degree to which the host processor is relieved of detailed supervision of data transfers, when the host

processor is managing communications with other systems via a bus,
it spends a great deal of its time waiting for events.

Thus, if a processor writes to its own local memory it requires
only a short address/data cycle, typically with no wait states.
However if it is writing to remote memory across the backplane it is
heavily wait-stated. When the address is recognized as non-local it
has to arbitrate for the backplane bus, put address and data on the
bus, hold the bus while the destination system effects the write and
signals "write complete" and write complete plus release bus has to
be signalled back to the source processor before it can terminate
the write cycle. Thus the processor is severely wait-stated when
accessing non-local memory because of the times taken for address
mapping, backplane bus arbitration, bus delays and the receiver bus
arbitration time (DMA to local processor). Such wait-stating
significantly reduces the bandwidth of the local processor bus.
Transfer of only a single cycle message on the backplane reduces bus
throughput by approximately 50% because of the inefficiency of bus
arbitration for each cycle and the use of block transfers with bus
hold. On the other hand large block transfers are more efficient
because there is no arbitration time between individual transfers on
account of bus hold, wasteful repetition of address information is
avoided and the local data bus bandwidth is improved - because data
movement takes place by more efficient DMA block transfers to or
from local memory, without extremely inefficient single cycle
processor wait-stating.

DMA transfers entail other disadvantages. A DMA controller is
needed and during the DMA process, local interrupts cannot be
serviced. Although the transfer is efficient, local bus latency is
increased. These problems can be overcome in part by using a
separate buffer memory for bus transfers, making block transfers
buffer-to-buffer. Nevertheless host processors still determine when
to send and receive and must initially exchange slow single cycle
messages. The host processor represents a performance bottleneck
because it has to manage the block transfers.

Further efforts to overcome the known problems involve use of
an intelligent block/message handler to relieve the host of much
management. For example a message-passing coprocessor is available

for Multibus II but such handlers operate with memory-to-memory
architecture and a typical backplane bus is quite poorly matched to
the message exchange requirements. The address bus is largely
redundant as are many clever, complex facilities (bus sizing, byte-
offsets, etc.). Interrupt facilities are not required. The bus
uses very many pins and has poor resilience in the areas of error
prevention, detection and tolerance, particularly in relation to the
vital matter of control.

The design of virtually all commercially successful buses is
rooted in the concept of a shared memory bus and as such,
arbitration, addressing, data and control functions are exercised
essentially in periods akin to memory cycle times, and
communications between multiple processors on the bus normally
employs a shared memory model.

Increasingly it is the bus bandwidth that is the limiting
factor in system design and wider buses (32 and even 64 bit buses)
are being employed to improve throughput - at some considerable
expense.

In most real-time system designs the programmer's preferred
model is based on the concept of an operating system in which
applications comprise a set of co-operating tasks, the ensemble
communicating with each other by means of messages (packets). When
this technique is extended to multi-processor systems - to increase
total processing power, modularity and resilience - messages have to
be passed from processor to processor. Increased modularity often
leads to increased message passing and again the shared memory (bus)
bandwidth ultimately limits the system performance.

There is yet another difficulty arising from the current shared
memory bus model in the case of faults and on-line enhancements or
reconfigurations of systems. Conventional system buses are not
designed to support such operations. Systems in general and data
communications systems in particular are becoming such fundamental
components of company infrastructures that non-stop operation is a
requirement. A new bus is called for that can provide solutions to
all of these problems.

Summary of the Invention

An object of the present invention is to meet this need and to provide a very fast inter-processor packet transfer service with very low host processor (engine) overhead, to maintain high integrity in the face of live board exchange, and to provide a more economical solution than that offered by current shared memory buses. As to speed it is an object to achieve high packet and information transfer rates without having to go to a high bus width.

The bus system according to the invention is defined in the appended claims and opens up possibilities for "networking" a set of modular intelligent processing 'engines', each 'engine' providing application specific functions to access various LAN, WAN (wide area network) or digital trunk services as well as providing general purpose processing, filing and packet switching capabilities.

Brief Description of the Drawings

The invention will be described in more detail, by way of example, with reference to the accompanying drawings, in which:

Fig. 1 is a block diagram of a system according to the invention with a plurality of engines connected via a packet bus,

Fig. 2 shows a conventional architecture for transfer of data to and from a micro processor back plane bus such as VME or Multibus,

Fig. 3 shows an embodiment of an improved architecture according to the invention,

Fig. 4 shows the structure of two VRAM buffer stores and the communication therebetween,

Fig. 5 shows the system of Fig.3 in more detail,

Fig. 6 shows a preferred embodiment of the invention,

Fig. 7 shows the addressing of data buffers by a host processor and a packet bus controller via a shared descriptor space,

Fig. 8 shows how descriptor chaining links a queue of buffers,

Fig. 9 shows how tasks on different engines communicate with each other,

Fig. 10 shows the structure of a buffer connection descriptor used to manage buffers,

Fig. 11 illustrates how buffers are managed in a connection list,

Fig. 12 illustrates how buffers are used as shared memory,   .

Fig. 13 shows the structure of a packet,

Fig. 14 shows the organization for transmitting packets to the packet bus,

Fig. 15 shows the organization for receiving packets from the packet bus, and

Fig. 16 shows a block diagram of a practical implementation of a system according to the invention.

<u>Detailed description of the Preferred Embodiments</u>

Fig. 1 shows one possible multi-processor architecture for a
system embodying the invention.   For purposes of illustration just
five computing "engines" 10 to 14 are shown.   Each engine may be
mounted on a separate printed circuit board in a rack mounted system
with a physically conventional backplane.   The backplane is used to
connect the engines 10 to 14 via a packet bus 15, which may have
only 16 information carrying lines.   Every engine incorporates a
packet bus controller 16 for communications via the packet bus.
Although just five engines are shown, there may be a much larger
number in a practical, complex system.

In the illustrated example the engine 10 is a master engine
which has supervisory control of the whole system.   This is a high
level mastership - the master is the engine which knows the
configuration of the system and maintains directories, lists of
connections, etc.   For system security the engine 11 is a shadow
engine which can take over from the master 10 in the event of
failure thereof.   The other engines are slave engines 12, 13 and
14.   This master/slave architecture applies only at the high,
supervisory level.   At the level of bus mastership all engines are
equal and any one engine can become bus master when it has won an
arbitration procedure so that mastership is passed from engine to
engine in accordance with their requirements for bus access.   Bus
arbitration can take place in a manner which is essentially
conventional.   Various arbitration procedures exist and it is
preferred to employ a procedure which ensures fair arbitration, with
no one engine able to shut others out of access to the bus.

Each engine typically interfaces also to at least one
peripheral controller.   Purely by way of example Fig.1 shows the
master engine 10 and shadow engine 11 interfaced to disc
controllers 17 and 18 respectively and the slave engine 12 connected
to an LAN coprocessor 19, e.g. the commercially available coprocessor
referred to in the introduction above.

The packet bus 15 is used to allow communication between
computing engines and between their various peripheral controllers.
However the packet bus 15 and the associated bus controllers 16 are
not constructed in the same way as conventional backplane buses and

associated controllers.

    Fig. 2 is a schematic illustration showing the architecture of
a known engine which communicates with other engines via a
conventional microprocessor backplane bus 20. The engine comprises
a host microprocessor 21 with its associated processor bus 22 which
connects the processor 21 inter alia to an input/output interface
device 23 (such as the disc controller), a backplane bus controller
24 and a DRAM buffer store 25 used to buffer data being transferred
between the processor bus 22 and the bus 20.

    For simplicity Fig. 2 does not show the RAM and ROM associated
with the processor 21. Such memory will be conventionally connected
to the processor bus 22. Data transfers to and from the bus 20 fall
essentially into two categories. Firstly the transfer may be from
or to a microprocessor task, i.e. from the RAM associated with the
microprocessor 21. Secondly the transfer may be from or to the
interface device 23 and thus from or to whatever peripheral is
served thereby on its external port 26.

    Fig. 2 illustrates schematically the operation of the known
system in the case of the latter alternative. Multiple DMA cycles
are required on the processor bus 22 to transfer a block of data
between the interface device 23 and the buffer store 25. Equally,
multiple DMA cycles are also required on the processor bus 22 to
transfer a block of data between the buffer store 25 and the bus
controller 24. Two DMA cycles are needed for every word transferred
in a double operation in this way between the interface device 23
and the controller 24, via the buffer store 25.

    Fig. 3 shows the essentials of an engine embodying the present
invention. The engine again comprises host microprocessor 21 (e.g.
an Intel 80C376), processor bus 22 and the I/O interface device 23.
The bus 20 is replaced by the packet bus 15 and the bus controller
24 is replaced by the packet bus controller 16. Furthermore the
DRAM buffer store 25 is replaced by a dual port RAM, preferably a
VRAM (video RAM) buffer store 30. In the case of a VRAM the first
port will be the conventional DRAM port and the second will be the
serial port connected to the SAM buffers described below. The first
port of the VRAM 30 is connected to the processor bus 22. The
second port is connected to a further bus 31 which allows direct

flow of data between the VRAM buffer store 30 and the packet bus
controller 16, without involving the processor bus 22. The packet
bus is 16 bits wide but the packet size may be varied up to the row
length of the VRAM (1kbyte for a 256k device). A serial access
memory (SAM) 32 is used for word serial/word parallel conversion to
enable the words of a packet to be sent or received, via the direct
data bus 31, 16 bits at a time.

        A bus controller conventionally manages data transfers in
accordance with various descriptors set up in the controller by the
host processor and in Fig. 3 block 33 (equivalent to Fig.5 block 41)
symbolises the descriptors utilized in the embodiment of the
invention. In accordance with the known architecture of Fig. 2, the
host processor 21 would set up I/O descriptors and bus controller
descriptors effectively to map the interface device 23 onto the
packet bus controller 16. RAM descriptors would also be set up to
control usage of the buffer store 30. Data would flow as in Fig.
2, with no use being made of the additional bus 31. However an
important feature of the apparatus resides in making the packet bus
controller 16 appear to the bus 31 similarly to the I/O
interface device 23. In other words the mapping is no longer
performed by the host through operations on the I/O and bus controller
descriptors but by effectively mapping the packet bus controller 16
itself onto the interface device 23. Having done this it is
possible to transfer data between the interface device 23 and the
packet bus controller 16 via the dual port VRAM buffer store 30 in a
completely transparent manner which requires no host processor
intervention. Accordingly, as shown in Fig.3, multiple DMA cycles
are still required on the processor bus 22 to handle transfers
between the interface device 23 and the first port of the VRAM
buffer store 30 but the processor bus 22 is not involved in the
transfers between the second port of the buffer store 30 and the
packet bus controller 16, via the additional bus 31.

        Fig. 4 shows the structure of two VRAM buffer stores and
symbolises the transfer of data from a first VRAM memory array 30A
via its SAM 32A and the packet bus 15 to a second SAM 32B and thence
to the associated VRAM memory array 30B. The packet bus
controllers 16 are not shown.

Each 256 k memory array consists of 16 bit-planes 35, each
having 512 rows and 512 columns. The memory array thus has a depth
of one 16 bit word, a width of 512 words i.e. one kbyte and a
capacity of 512 rows, each of one kbyte. Data transfers are
processed in units of one such row and a packet can occupy the whole
or part of a row. The buffer descriptors define single buffers for
general use and transmit and receive between VRAM ports. Each of
these buffers consists of one VRAM row. The descriptors 33 are used
to set up logical connections between devices and tasks, each
connection having none, one or more buffers.

For simplicity Fig. 4 does not show the first ports of the VRAM
memory arrays 30A and 30B but it will be realised that these memory
arrays are entirely normal, commercially available devices.
Symbolic connections 36A and 36B represent the parallel transfer of
a complete buffer row, in the case of the memory array 30A from a
representative row 37A to the associated SAM 32A and, in the case of
the memory array 35, from the SAM 32B to a representative buffer row
37B.

Each SAM 32A, 32B is a one kbyte static RAM, capable of holding
one buffer row.   In the case of the transfer symbolised in Fig. 4,
the SAM 32A accepts a complete 1kbyte row from the VRAM memory array
30A in a single read operation.   The words are then read out of
the SAM 32A serially and, after transfer via the packet bus 15, they
are read 16 bits at a time into the SAM 32B.   The complete 1kbyte
row thus assembled in the SAM 32B is then written in one operation
36B into the VRAM memory array 30B.

There is no concept of separate receive and transmit FIFO
buffers as commonly found in VLSI communication controllers.   These
are replaced by the SAM of the VRAM which is shared by both the
receive and transmit channels.

Rapid resolution of SAM read and write functions (into and out
of the VRAM array) is required in order to avoid unnecessarily slow
usage of the SAM buffer.   Obviously, whilst the SAM contains
received data which has not yet been transferred to the VRAM array
it cannot be used to transmit data.   The reverse is also true.
To this end the packet bus controller 16 will expect that a row
destination address has been located in a pointer register on the

processor bus 22 for receive operations and that a similar register
shall hold a row source address pointer for transmit operations.
At the appropriate time the packet bus controller 16 requests DMA
access to the host memory bus and when a grant has been received
enables the appropriate pointer register and VRAM control lines to
achieve the required read or write operation between the SAM and the
VRAM row.

The SAM, which is shared by both the receive and transmit
channels, may also be used to copy data (buffer copy) between VRAM
rows.   The row to be copied becomes the transmit row and the row
which is to contain the copy becomes the receive row.   A simple
instruction to the packet bus controller can accomplish the copying
operation in a few machine cycles.   No transfer takes place on the
Packet Bus 15.

Parallel/serial and serial/parallel conversions of this nature
are extremely well known in the art and further description of the
SAMs will not be given.

Although the packet bus 15 is only 16 information bits wide,
with the architecture of Fig. 3 it is possible within the limits of
readily engineered technology to achieve a transfer rate of 20 MHz
which with 16 bit words is 40Mbyte/s.

The system according to the invention as shown in Fig.3 will
now be described in more detail. Fig. 5 shows the host processor 21
with its bus 22, the VRAM 30 and SAM 32, the I/O interface device or
peripheral controller 23 and the packet bus 15, all as in Fig.3.
The remaining items of Fig. 5 represent the packet bus controller 16
in more detail plus the memory serving to store the descriptors 33.
Referring firstly then to the memory, the processor bus 22 is a
typical computer bus carrying data, addresses and various control
signals. Purely by way of example, the host processor 21 could be an
80186 processor, in which case the bus 22 is an 80186 bus.
Associated with the VRAM 30 is further RAM providing code space 40
for the processor and space 41 for buffer descriptors, i.e. data
describing the buffers set up within the VRAM 30.   The host
processor 21 can perform conventional read and write accesses to the
RAM 40, 41 and the VRAM memory space 30 although this is managed by
the packet bus controller 16.   The host processor 21 can set up the

way in which the packet bus controller effects the RAM management.
The host processor 21 has no control over memory accesses via the
SAM 32. A conventional DMA controller 43 is provided for managing
data reads and writes between the bus 22 and the memory 30, 40, 41,
specifically for efficient transfer of descriptors between the host
and RAM. The DMA controller 43 handles I/O peripheral and packet
bus transfers to and from the memory, which is always addressed on
its first port via the controller 43, as shown by the bus
connection 22A.

The heart of the packet bus controller 16 is a buffer manager
or control processor 44, which might be a 320C26 or a 320C25 chip,
for example. This has its own data RAM 45 and code ROM 46. A
clock signal is input from the packet bus 15 via a synchronising
circuit 47.

The control processor 44 can communicate with the host
processor 21 via the processor bus 22 and a host-system interface
comprising an interface and control unit 48 which includes custom
control logic, conventionally implemented as a programmed logic
array. Thus the control unit 48 controls the DMA controller 43
when DMA data transfers of e.g. descriptors are being effected
between the VRAM 30 and the processor bus 22. Transfers between the
VRAM 30 and the I/O interface device 23 will be handled by the DMA
controller conventionally included in this device 23.

A word bus 50 (corresponding to the bus 31 of Fig. 3) is
connected to the SAM 32 and is connected to the control processor 44
through a packet handler 49 which includes the bus arbitration
logic. The word bus 50 is further connected to the packet bus 15
through driver transceivers 51, such as conventional Futurebus
transceivers. These are controlled by the packet handler 49. The
packet handler 49 is preferably constructed as a state machine (i.e.
in hard-wired logic). The hardware arbitration unit in the packet
handler 49 can be constructed in the same manner as the known
Futurebus arbitration unit.

In a preferred embodiment of the invention as shown in Fig. 6,
the host-system interface comprises a dual-port SRAM 52 which forms
the descriptor memory. The SRAM ports are connected to the host
processor bus 22 and to the control processor or buffer manager 44.

Since the buffer space, being a large consumer of memory, is
entirely located in the VRAM there is relatively little need for a
large shared memory area between the host 21 and the packet bus
controller 16.   The dual-port SRAM may be accessed simultaneously
by both the host and packet bus controller making it an ideal
location for shared buffer descriptors and message passing
mailboxes.   The addressing of the descriptor space and the VRAM
data buffers is shown in Fig. 7.

   The descriptor space may not merely contain pointers to a VRAM
buffer but may also describe the particulars of the data which it
represents.   This information is required to successfully route
data from one packet bus controller to another across the packet
bus.

   An important advantage of descriptor operation in a shared
memory space is the ability to chain descriptors together to form
queues as shown in Fig. 8.   Chaining allows one processor to add
descriptors to one end of the queue whilst another processor removes
descriptors from the other end.   Descriptor to descriptor
pointers are thus required and flags will be needed to indicate
when the end of a queue has been reached.   The goal is to achieve
dynamic queueing with minimal interruption of the processors acting
on the queues.   The processors are of course in this case the
buffer manager and the host processor.

   The concept of frame fragmentation (as found in many ethernet
controllers) over several buffers is not supported.   A frame will
always be less than or equal to the maximum size of the VRAM row.

   Similarly, buffer fragmentation is not allowed.   A VRAM buffer
is always of a size equivalent to the VRAM row size.   The buffer
may or may not be filled with data.   The amount of data contained
in the buffer is indicated in the data description area of the

   The buffer manager or control processor 44 administers and co-
ordinates the exchange of data between the host system and the
packet bus.   The host need not perform these relatively simple but
time consuming tasks.   In order that the buffer manager can
operate independently from the host it is located on its own bus,
the packet bus controller control bus, which is isolated from the
host bus by the host-system interface or the dual-port SRAM.

In the implementation in Fig. 5 the interface 48 affords the
buffer manager 44 a window into the host memory space through which
it may directly access data structures contained in the host memory
space.   This then is a shared memory arrangement and is the
preferred solution for most high performance VLSI communications
controllers.   Such devices access host memory space by means of
Direct Memory Access.   No latency is introduced by the interposing
of the buffer manager between the host and the packet bus firstly
because of the efficiency of the shared memory arrangement, which
allows the buffer manager and host to operate on the same data
structures, and secondly because the reduction in processing latency
due to the reduced host workload more than offsets the latency
caused directly by interposing the buffer manager.

When the shared DMA accessed memory (in Fig. 5) is replaced with
dual-port SRAM (as in Fig. 6) a further increase in performance may
be realised since the buffer manager and the host processor can
access shared memory structures at the same time.   This is
particularly attractive because no data passes across this
interface.   The data goes by way of the VRAM.   Therefore the
expensive and relatively low density dual-port SRAM need only be big
enough to support the controlling structures or descriptors.

The systems of Figs. 5 and 6 operate as follows.
The configuration of the bus controller 16 is established by the
host processor 21. To this end it commands the control processor or
buffer manager 44 through the interface and control unit 48 or the
dual port SRAM 52 and sets up buffer descriptors and higher level
connection descriptor within the descriptors area 41 or the
SRAM 52.

The buffer descriptors describe single buffers, each consisting
of one word-wide VRAM row. The connection or queue descriptors
describe a list of buffers and can consist of none or more buffers.
Once created the buffers are referred to by the first address and
are taken to be a uniform size of 1kbyte. Buffers can be assigned
for two purposes. Some are used by the host processor for task to
task communication. Others are assigned for communication with the
I/O device 23.

The assignment of the queues and the mode of communication
therewith is determined by the high level descriptors set up in the
memory space 41 or 52 by the host processor 21.

Dealing firstly with the task queues, the configuration thus
established is illustrated schematically in Fig. 9 which shows two
host engines 60A and 60B communicating via their respective packet
bus controllers 16A and 16B and the packet bus 15. Although the
packet bus 15 comprises only one physical connection, conceptually
it provides a plurality of separate logical connections 15A, 15B,
15C in the illustrated example.

Each host engine 60A, 60B runs a plurality of tasks illustrated
as 60A1 to 60A6 and 60B1 to 60B6. By way of example it is assumed
that task 60A1 is to communicate with 60B2 via the logical
connection 15A, the task 60A5 is to communicate with the task 60B3
via the logical connection 15B and the task 60A6 is to communicate
with the task 60B5 via the logical connection 15C.

Within each packet bus controller 16A, 16B there are a
plurality of queues designated Q1 to Q64 in each case. Each of
these queues comprises one or more linked buffers. 64 queues are
shown by way of an example. A queue can be empty, that is it may
have no associated buffers. It will be recalled that the VRAM 30
has a capacity of 512 rows so the maximum possible number of queues
is 512 or, given duplex links each using two queues, 256 if each has
only one row. Each queue of Fig. 9 can comprise one or more rows ,
but other rows must be left free to implement the communications
with the I/O interface device 23, as described below.

In Fig. 9 is it assumed that queue 2 in the packet bus
controller 16A is assigned to the task 60A1 while the queue 1 in the
packet bus controller 16B is assigned to the task 60B2.

As is conventional in packet communication each packet includes
address information, including the destination address. In
implementing the scheme according to Fig. 9, this address
information includes not only the address of the destination engine
but the address of the queue within that engine for which the packet
is destined. This address is received by the packet handler 49 and
passed to the control processor or buffer manager 44 of a
destination engine which selects the correct queue within the

associated VRAM 30 to receive the packet.

So far as communication with the I/O interface device 23 is
concerned, an important feature of the system resides in the high
level descriptors set up within the memory space 41 or 52. The I/O
interface device 23 will have its own well defined communications
protocol, e.g. as well understood in relation to disc controllers
and the 82596 co-processor already referred to above as examples of
devices 23. The descriptors set up in the space 41 or 52 mirror the
device 23 as it appears to the bus 22 so that entirely transparent
communication is possible between the I/O interface device 23 and
the packet bus 15 via the VRAM 30.   Although further details of
this aspect of the system are given below it should be pointed
out here that its implementation does not involve fundamental
problems. In accordance with conventional practice, it is necessary
to write a software driver for the host processor 21 in order to
match the control protocol and registers of a peripheral controller
such as the I/O interface device 23. In effect, this driver is
transferred from the host processor to the bus controller 16 so that
the device 23 can communicate with the bus 15 via the VRAM 30
without any need for host processor supervision.   To effect a
transfer the host processor simply issues the top level descriptors
therefor and then leaves the controller 16 to manage the transfer.

As a further facility, the control processor 44 can force row-
to-row data transfers within the VRAM 30, even although the data
does not leave the memory chip.   This provides additional
flexibility in setting up logical connections.

The 320C26 processor preferably utilised for the control
processor 44 has a 1K word internal memory space and the whole
coding necessary can be fitted into this memory space in assembly
code, thus promoting very high speed operation.

The code provides all buffer management functions in addition
to the overall control of the packet bus subsystem. The control
processor or buffer manager 44 controls all allocation and de-
allocation of buffers within the VRAM 30 memory space and, although
the host processor 21 is given the buffers to use, ultimately
ownership of the buffers always reverts to this processor. An
alternative to this rule occurs when an external I/O device 23 is

attached, which has in-built its own specific buffer management
methods.   In this situation the I/O device often manages the use of
buffer descriptors and the buffers themselves.   Such a case is
described in a later section.   In both cases similar data structures
and connection set-up procedures are used, the main difference lies
in the sole or shared management of these structures, in the host
task to task case and the I/O processors support case respectively.

     The following describes the case of the host task to task
buffer management scheme assuming that the I/O device 23 does not
provide nor requires any buffer management.   Here the packet bus
controller speeds up packet transfer by relieving the host of this
task.

     The initial situation after a reset is that all the buffer
pointers are linked in a large linear list.   This list can be
linked back on itself to create a circular list when this is
expedient.   All list pointers are either stored in shared SRAM 52 or
are stored within the buffer descriptor space 41 in host memory but
are accessed by the control processor 44 as a shared resource.

Data Structure Profiling

     Data structure profiling is a means by which the buffer manager
operates directly on the descriptor formats of another device.
This avoids the time consuming complexities of converting from one
descriptor structure to another as data passes between an external
port utilising a VLSI controller and the packet bus controller.

     The descriptor root pointers inside the VLSI controller may be
set up so as to locate its data structures in the small dual-port
SRAM which forms the interface between the controller and the buffer
manager.   The dual-port nature of this RAM allows these data
structures to be parsed and modified directly by the buffer manager.

Connection Setup

     The host can request the creation of a number of connections to
tasks both on distant cards (engines) and on the same card.   The
maximum number of simplex connections is restricted to 512 for
practical reasons;   however fewer would be used in practice.   The
initial set up procedure creates a data structure for each
connection requested by the host which contains pointers to the
associated buffer queue and size limitation data.   The queue

pointers are initially NULL as there are no associated buffers. The
connection is also not aware of the destination address as this is
supplied by another mechanism once the connections at both ends are
active. The structure of a connection descriptor is illustrated in
Figure 10.

Buffer Allocation

When a buffer allocation request is made by the host processor
21, the first available buffer is removed from the list of free
buffers. The free buffer list operates on the first in - last
out principle. The free pointer is modified to point to the next
free buffer and a value is placed in the allocated buffer pointer
to indicate that it is no longer the property of the control
processor 44. The buffer stays the property of the host until such
time as it is relinquished. When relinquished the buffer is returned
to the free list as the next available free buffer. Whilst the
property of the host processor 21 the buffer can be used for any
purpose and responsibility for avoiding corruption lies with the
host. Figure 11 shows the buffer management structure once some
connections are active.

Packet Transmission

When the host has been allocated a buffer and has assembled
data within this buffer which it wishes to be transmitted by the
packet bus, it requests a send. The buffer then becomes part of
the queue of the specified connection. The connection queue is
circular and is a first in - first out queue. Head and tail
pointers identify the front and rear of the queue and a free/used
word is used to determine the state of the queue. The last word of
the connection data block identifies the destination card and
connection.

Once transmitted successfully the buffer is returned to the
free list as indicated above.

Packet Reception

When an incoming packet is detected, the packet handler 49
requests the allocation of a buffer from the free list. This
operation is the same as for the host above. If the packet is
correctly received this buffer is appended to the connection queue
for collection by the host.

The host makes a request to receive a packet from a specified connection and this causes the first buffer to be removed from the queue and to be treated as allocated to the host. Once the host has finished with the buffer and its data it returns the buffer to the free list in the usual way.

Buffer Copy

This is effectively an internal packet transmission only without the specification of a connection. It can be performed non-destructively on any buffer within the VRAM space.

General Packet Transfer Process

The transmit procedure is activated when arbitration 80 (Fig. 13) is won on a card. The packet handler 49 has assembled a header block 81 for the appropriate data packet. The actual packet sent is determined by a sequencing procedure performed before bus access is requested. This procedure checks each connection block for packets to be transmitted. The header block terminates with a checksum 82.

The header is transmitted as a broadcast although only one card is expected to respond. If no response is forthcoming within a specified period (time out) then packet transfer is abandoned. This is also the action taken if the receiver reply 83 indicates that transfer is unacceptable. If there is a good reply the packet data 84 is sent out onto the bus 15 followed by the checksum 85. The transmitter again expects a reply 86 at the end of this and, given that all appears well, it will remove the buffer from the transmit queue and place it on the free list. Fig. 13 is not drawn to scale on the time axis.

To ensure that a packet is not sent twice in error a sequence bit is contained in the header 81. The state of this bit is inverted when the packet is good. Thus if the transmitter believes the packet bad and the receiver thinks it is good the sequence bit gets out of step and is detected as such when the packet is re-transmitted. Only one bit is required as the system cannot get more than one packet out of step.

The participating cards are synchronized prior to the header transmission so that all further interaction occurs correctly.

The receive activity on non-transmitting cards is initiated after arbitration has taken place. All non-transmitting cards are

expected to listen for an incoming packet which comprises a header
block 81 followed by the packet data.

The header is a six word block of data which contains the
destination address, source address, packet length and a control
word. The final two words are the checksum 82 to ensure header
integrity.   If the checksum fails then the procedure is terminated
immediately whereas if it passes, the header is examined to find the
destination address. Given that the destination matches the
procedure continues towards packet reception. Any irregularity in
the header will cause a receiver to abandon the transfer; thus only
one card will respond to any header with a reply 83.

The connection specified by the number extracted from the
header is prepared for receipt of data. If the connection is unable
to accept the data the transmitter is informed (reply 83) and
should relinquish the bus and try again later. Given that the
header is correct and the reply to the transmitter is favourable
then packet data transfer commences. Data is shifted out of the
SAMs on the transmitter and into the SAMs on the receiver. At both
ends the packet handler 49 is reading the data as it appears on
the bus and is calculating a checksum. As soon as the data transfer
is complete the sender's checksum 85 is transmitted and is compared
at the receiver. The transmitter is immediately informed of the
result of this check 86 and will act accordingly. The receiver will
abandon any packet that fails the checksum and will only insert good
packets into the appropriate connection queues.

Further consideration will now be given to the specific example
where the I/O interface device 23 is the aforementioned 82596 LAN
co-processor.   Fig. 12 shows how the shared memory structure of the
VRAM 30 is managed in this case.   The drawing also shows the
initialization root 70 and the main system control code block 71.
Once the buffer size has been determined the host processor 21
creates the buffer descriptors for both the receive and transmit
lists.   The buffer descriptors are chained in their respective
lists and the proportion of transmit to receive buffers is
determined by the application of the engine.

The transmit and receive operations will finally be considered
in more detail, dealing with transmission to the packet bus first.

Transmit to Packet Bus

Referring to Fig. 14, a data frame received by the peripheral
controller 23 from the external data link 26 is placed into one or
more chained receive buffers 90. A Frame Descriptor (FD) 91 is
constructed containing such information as length, source and
destination address. This information is directly available to the
packet bus controller and is used to derive the connection details
for onward transmission, The connection block in the control
processor or buffer manager 44 memory pertaining to the specified
destination address contains the destination card and connection
number allocated by the master card in the system. This is used to
generate the packet header.

The peripheral controller may accept frames for only one
destination address or, alternatively for a group of destinations.
The packet bus controller must determine which transmit connection
it should use for packet bus transmission. The onward connection
number is derived from the destination address using a form of
address translation. This translation is achieved using either a
software lookup table or a Content Addressable Memory in hardware.
It should be noted that on the remote packet bus card the opposite
translation must be applied to restore the original address.
Translate tables are constructed when the system is initially set
up.

Frames are re-transmitted on the packet bus in the order that
they are received by the peripheral controller and, where a frame
extends over more than one buffer, a continuation bit in the packet
header will ensure correct re-assembly at the receiver.

Fig. 14 illustrates the procedure outlined above. Data
received by the peripheral controller is placed into receive
buffers 90 from whence it is extracted by the packet bus controller
which references the frame descriptor.

The frame descriptors 91 are arranged in a loop configuration
(as are the buffer descriptors (BD) 92) with the last unused frame
marked. Once the data has been sent on by the packet bus the frame
descriptor and its associated buffer(s) are returned for use by the
peripheral controller. This will be achieved by manipulating the
End Of List bit in the free buffer list. This would allow the list

to be made circular, simplifying much of the buffer manipulation.

The packet bus controller 16 examines the frame descriptors FD for new data and removes data directly from the receive (from peripheral controller) buffers 90, putting the data on to the packet bus 15, without any new multi-cycle DMA operation.

Receive from Packet Bus

While the peripheral controller 23 controls the buffers which receive data from this controller, the packet bus controller 16 controls the buffers which receive data from the bus 50 (Fig. 15). Where the first buffers are well ordered, the second buffers are less so. This is because, although packets arrive in the correct order where multi-buffer frames are concerned, the actual sequence of received buffers may be made up of more than one frame. That is, buffers of one frame may be interspersed with those of another frame. The packet bus controller must assemble the entire frame before instructing the peripheral controller to transmit the data.

When the packet bus receives a packet the data is placed into the first available buffer 95. If the continuation bit in the header is not set indicating that this is the only buffer of a frame then the packet bus controller 16 performs address translation, assembles a transmit command TC and places this into the command queue. When the packet bus receives a packet with the continuation bit set a different course of action is required. The header does not specify the length of the frame in buffers and so assembly of the frame requires that the buffers be taken from the free list as required and added to any buffers already associated with the connection. If a packet for another connection arrives between the first and last buffer of the first connection it will take the next buffer available out of the list. The packet bus controller 16 will join the buffers to assemble the full frame before creating the transmit command and adding this to the command queue.

Each receive connection (in the packet bus) holds a temporary queue of buffers until the frame is complete. The entire queue is passed to the host as a transmit command which points to the first buffer descriptor of the frame.

The peripheral controller 23 examines the command list and acts on any commands (TC or CB = command block) found therein. On

finding a transmit command TC, the controller 23 will remove data
from the transmit (to peripheral) buffers 95 and will assemble
and transmit a frame.

Practical Implementation

A practical implementation of a system according to the
invention is shown in Fig. 16.   The packet handler ASIC is married
to a buffer manager processor (Texas Instruments TMS320C25) and
sub-system and interfaces to a host processor (Intel 80C376) and
sub-system through dual-port SRAM for control purposes (descriptors
and interprocessor messages) and through dual port VRAM for data
exchange.   Fig. 16 provides full details of the components therein
and will be understood by the man skilled in the art in the light
of the description given above of the foregoing figures.   A full
textual description of Fig. 16 is not therefore necessary here.

CLAIMS

1.   A computer bus system comprising:

a host processor;

a local processor bus associated with the host processor;

a packet bus controller having a port adapted for connection to a communications bus;

a host-system interface associated with the packet bus controller and coupled to the host processor for controlling the transfers which are to be effected by the packet bus controller;

a dual-port RAM having first and second ports, the first port being connected to the processor bus; and

coupling means coupling the second port of the RAM to the packet bus controller;

the packet bus controller being adapted to transfer data through the coupling means directly between the RAM and the communications bus without intervening control by the host processor.

2.   A system according to claim 1, wherein the RAM is VRAM.

3.   A system according to claim 1, wherein the coupling means is a direct data bus.

4.   A system according to claim 1, wherein data transfers between the RAM and the communications bus take place in units of up to one row of the RAM.

5.   A system according to claim 4, wherein the transfers are effected via a serial access memory which converts a row of data (e.g. 1024 bytes) into a succession of words having a width (e.g. 16 bits) equal to the width of the packet bus.

6.   A system according to claim 1, further comprising a peripheral controller connected to the processor bus and wherein the packet bus controller is adapted to transfer data directly between the

peripheral controller and the communications bus via the RAM, serving as a buffer, and the coupling means.

7.   A system according to claim 6, wherein the packet bus controller is, at descriptor level, configured to act directly on the control structures of the peripheral controllers.

8.   A system according to claim 7, wherein the packet bus controller is software configured under control of the host processor.

9.   A system according to claim 1, wherein the packet bus controller is adapted to set up a plurality of buffers in the RAM to enable functionally-separate logical connections between the communications bus on the one hand and, on the other hand, separate tasks on the host processor and/or one or more peripheral controllers.

10. A system according to claim 9, comprising a plurality of engines connected to the communications bus and each comprising a host processor, local processor bus, RAM, and packet bus controller as in claim 1, wherein buffer descriptors stored in different engines set up functionally-separate logical connections between tasks running on different host processors.

11. A system according to claim 1, in which the host-system interface comprises a direct memory access (DMA) controller for accessing a descriptor memory.

12. A system according to claim 11, in which the descriptor memory is in host memory.

13. A system according to claim 1, in which the host-system interface comprises a dual-port SRAM having a first port connected to the processor bus and a second port connected to the packet bus controller.

FIG.1

PRIOR ART   FIG.2

FIG.3

SUBSTITUTE SHEET

3/11
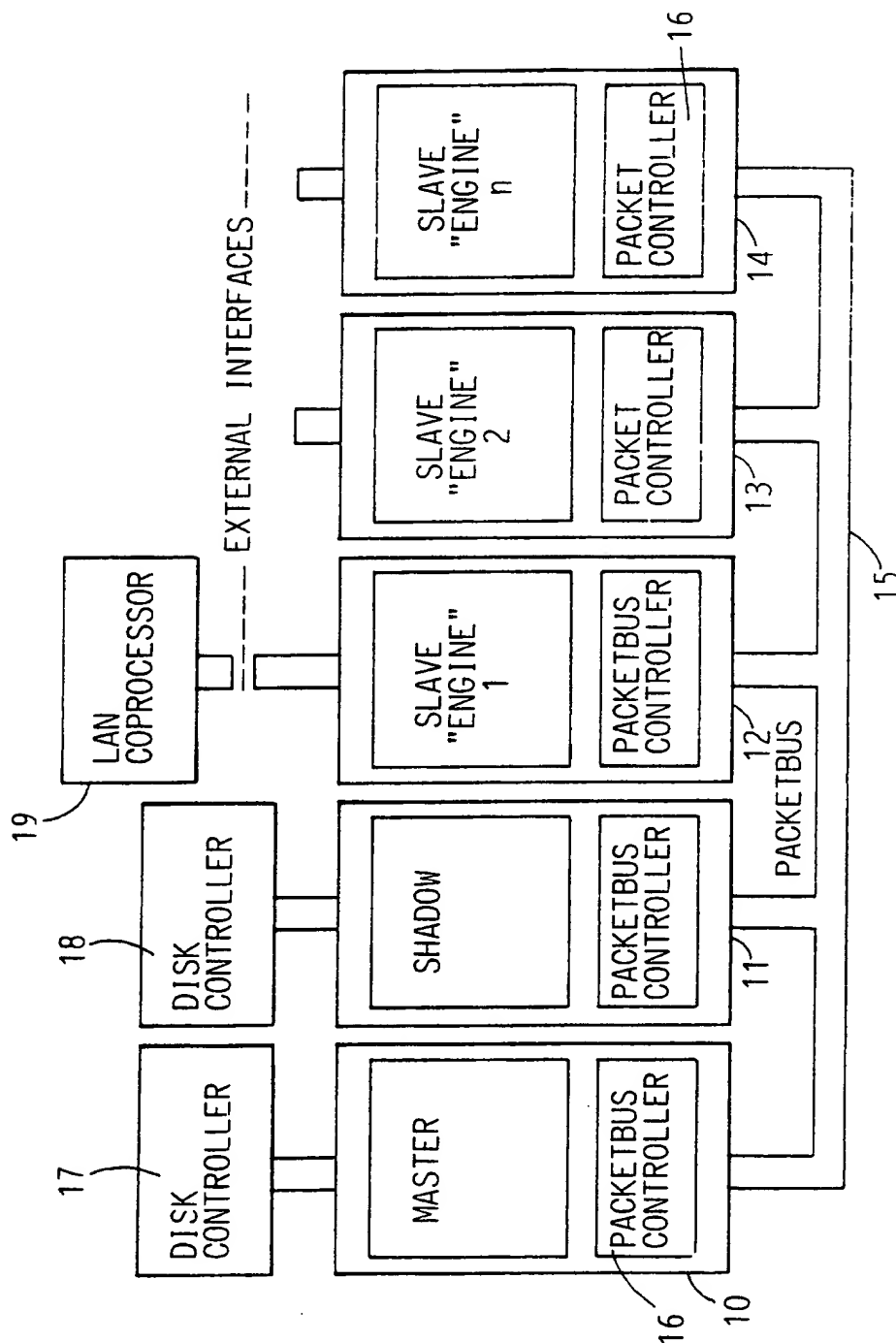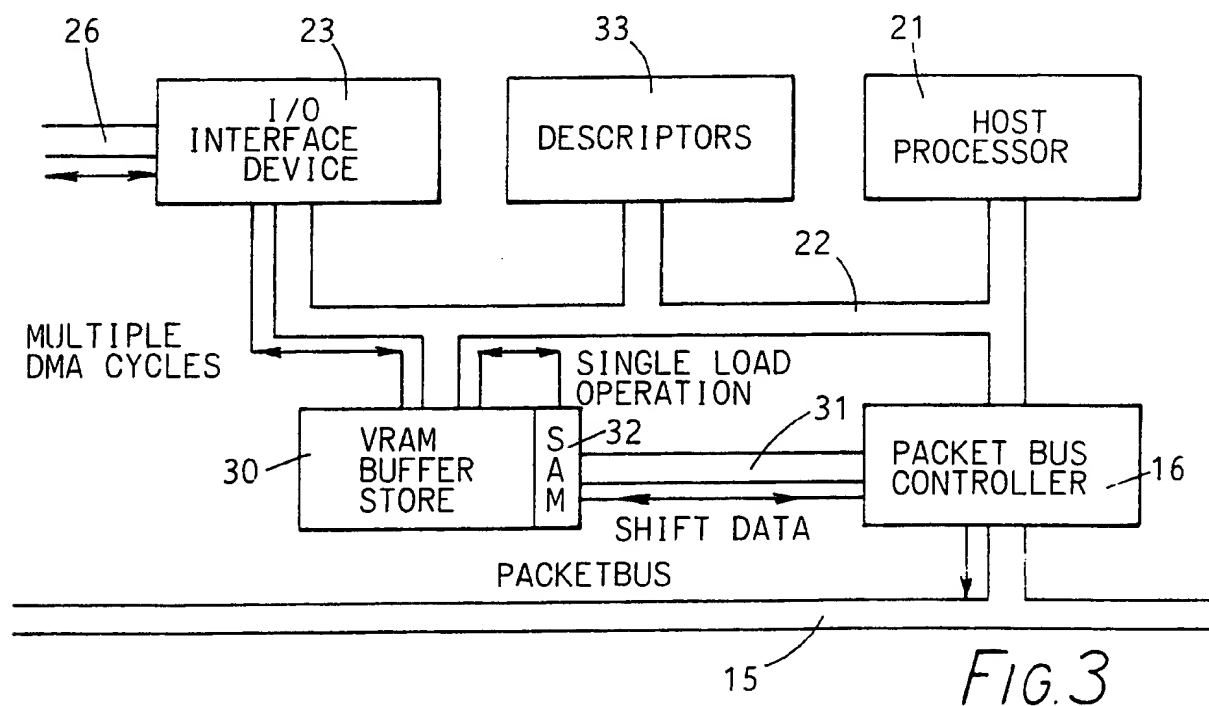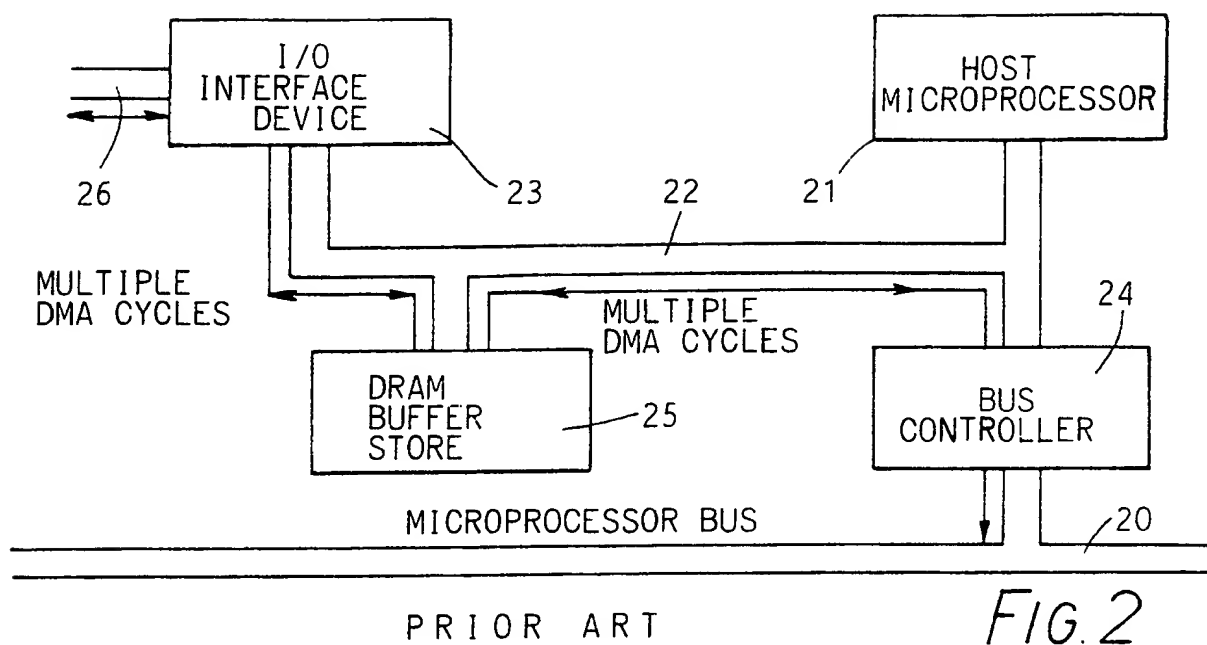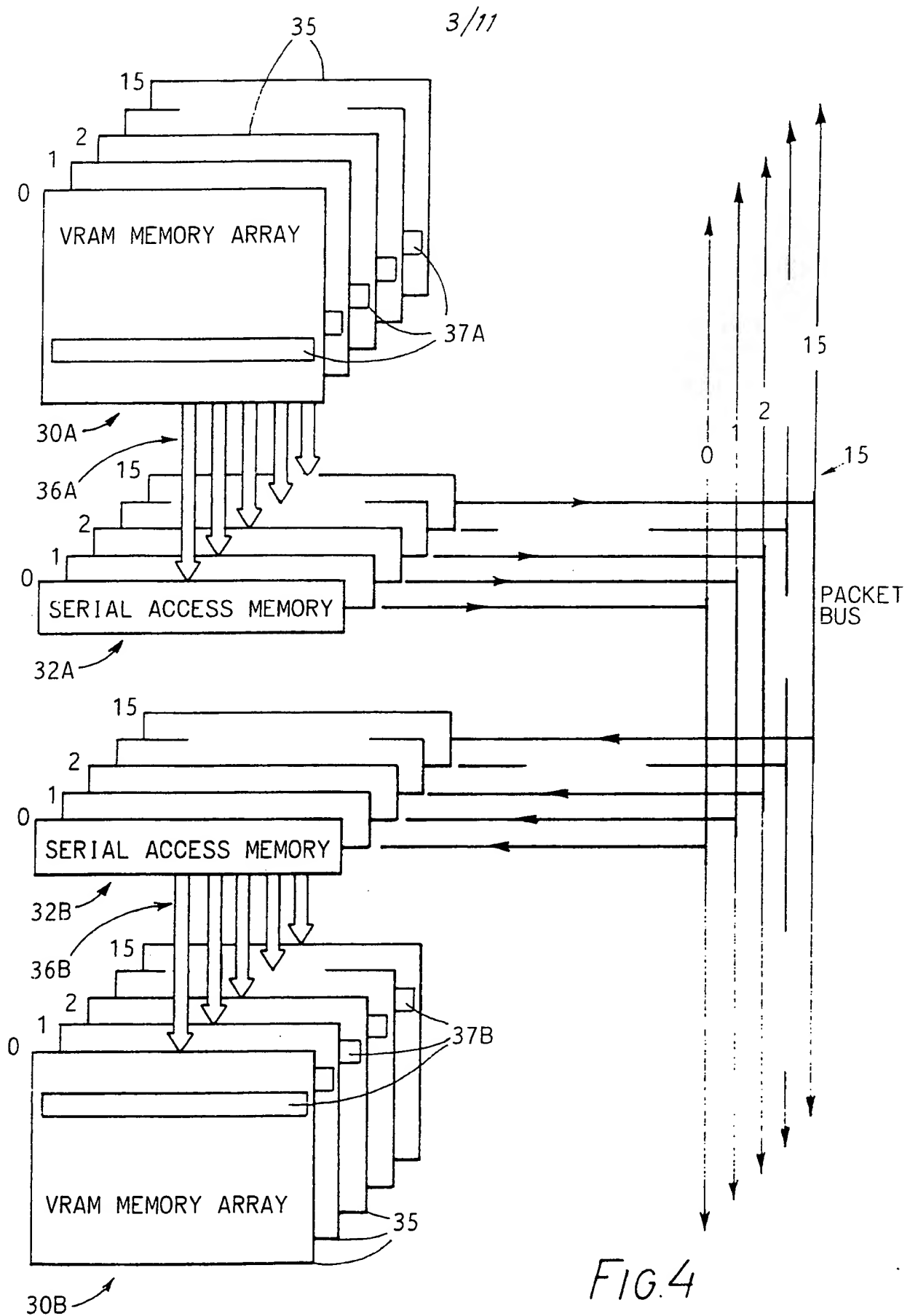
FIG.4

SUBSTITUTE SHEET

4/11

FIG. 5

5/11

HOST MEMORY (DRAM)   HOST PROCESSOR   21   BUFFER MEMORY (DUAL-PORT VRAM)   30

22

PERIPHERAL CONTROLLER   DESCRIPTOR MEMORY (DUAL-PORT SRAM)   52

23

26

BUFFER MANAGER (BM)   44

-16-

49   PACKET HANDLER (PH)   31

PACKET BUS INTERFACE (PBI)   51

15

PACKET BUS

FIG. 6

DRAM

HOST PROGRAM SPACE

DESCRIPTOR PTR

HOST ADDRESS SPACE

VRAM

VRAM DATA BUFFER

DUAL-PORT SRAM

DESCRIPTOR SPACE

HOST AND PBC SHARED ADDRESS SPACE

VRAM DATA PTR

SMALL SRAM

PBC PROGRAM SPACE

DESCRIPTOR PTR

PBC ADDRESS SPACE

FIG. 7

SUBSTITUTE SHEET

FIG.8

FIG.10

	PRIMARY USE
	HEAD POINTER
	TAIL POINTER
	BUFFERS USED/BUFFERS FREE
	DESTINATION ADDRESS

BIT NUMBER

F E D C B A 9 8 7 6 5 4 3 2 1 0

CARD NUMBER
CONNECTION NUMBER
DEALLOCATION MARKER BIT
SEQUENCE INTEGRITY BIT
TRANSMIT CONNECTION IDENTIFIER

1
2
3
4

FIG. 9

8/11

FREE BUFFER
LIST

CONNECTION
LIST

CONNECTION
QUEUES

FREE →

UP TO 509
BUFFERS

CONNECTION 1 TX

HEAD 1	
TAIL 1	
8	L1-8
DEST ADDRESS	

CONNECTION N TX

NUL PTR	
NUL PTR	
0	LN
DEST ADDRESS	

EMPTY CONNECTION

CONNECTION 31 RX

HEAD 31	
TAIL 31	
1	L31-1

FIG. 11

SUBSTITUTE SHEET
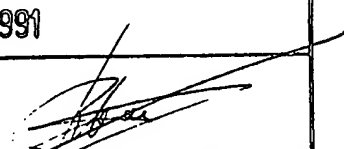
9/11

FIG. 12

FIG.13

BUSY

10/11

80 ARBITRATION

81 82 HEADER TRANSFER

83 REPLY

84 PACKET DATA

85 CHECKSUM

86 REPLY

T=0 (BUSY LINE BECOMES ACTIVE)

T=n (START OF NEXT PACKET)

FIG.14

ORDER OF RECEIVE

PERIPHERAL CONTROLLER

26

23

DATA IN

1 FD 91   2 FD 91

BD   BD 92   BD

90   90

1   2   3

ORDER OF TRANSMIT

DATA OUT (ON TO PACKET BUS)

50

16 PACKET BUS CONTROLLER

FIG.15

ORDER OF EXECUTION

PERIPHERAL CONTROLLER

26

23

DATA OUT

1 TC   2 CB   3 TC

BD   BD 92   BD

95

1   2   3

ORDER OF PACKET RECEIVE

DATA IN (FROM PACKET BUS)

50

16 PACKET BUS CONTROLLER

SUBSTITUTE SHEET

11/11

HOST DATA BUS<0..15> (32 BIT PIPELINED)

FIG.16

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)		
Category°	Citation of Document, with indication, where appropriate, of the relevant passages	Relevant to Claim No.
A	EP,A,0234181 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 02 September 1987 see the whole document ---	1, 2

ANNEX TO THE INTERNATIONAL SEARCH REPORT
ON INTERNATIONAL PATENT APPLICATION NO.

GB 9100035
SA 43509

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

14/03/91

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP-A-0234181	02-09-87	JP-A- 62184559	12-08-87